

You Get What You Pay For—Except You Don’t: Instance Families and Scaling Strategies for Cloud Video Transcoding

Breno José Ribeiro de
Vasconcelos
bjrv@cin.ufpe.br
Centro de Informática,
Universidade Federal de
Pernambuco (UFPE)
Recife, Pernambuco, Brazil

Elisa de Fátima Andrade
Soares
efas@cin.ufpe.br
Centro de Informática,
Universidade Federal de
Pernambuco (UFPE)
Recife, Pernambuco, Brazil

Carlos André Guimarães
Ferraz
cagf@cin.ufpe.br
Centro de Informática,
Universidade Federal de
Pernambuco (UFPE)
Recife, Pernambuco, Brazil

ABSTRACT

Cloud video transcoding is widely deployed in video-as-a-service pipelines, yet practitioners lack statistically grounded guidance for selecting and scheduling EC2 instance types under real cost constraints. Conventional wisdom recommends compute-optimized instances for encoding workloads, but this ignores provisioning dynamics that can dominate wall-clock time and expenditure. Existing benchmarks compare instance families without controlling for provisioning latency or scaling strategy, leaving the true cost driver unidentified. We address this gap through a large-scale controlled experiment spanning more than 5,700 trials across 15 instance types, three codecs (H.264, H.265, VP9), and three scaling strategies, with 30 repetitions per configuration and 95% confidence intervals throughout. Our methodology decouples provisioning from encoding time and evaluates custom-AMI pre-warming, horizontal scaling, and chunk-size sensitivity across resolutions. We show that provisioning latency, not CPU encoding, is the dominant cost factor: naïve horizontal scaling lets provisioning consume up to 93.3% of wall-clock time, whereas a pre-warmed 10-node `t3.micro` cluster outperforms a single `m5.4xlarge` by $1.28\times$ at 89.4% lower compute cost. These findings are translated into concrete, statistically grounded guidelines for sizing and scheduling cloud transcoding clusters.

KEYWORDS

Cloud Computing; Video-as-a-Service; Cost Optimization; Video Transcoding; Horizontal Scaling; Provisioning Overhead; Elasticity; ARM Graviton

1 INTRODUCTION

Video transcoding is one of the most CPU-hungry tasks in media delivery, and cloud platforms are where most of it runs today. For Video-as-a-Service (VaaS) operators, the EC2 instance family they choose has a direct impact on margins [16, 18]. The usual advice is to reach for compute-optimized instances [6] when encoding is the bottleneck, on

the assumption that faster hardware means lower cost per job. In practice, the picture is more complex: under the burst conditions and codecs evaluated, burstable instances proved up to $29\times$ more cost-efficient despite being slower in absolute terms. Yet this advantage diminishes at scale, not because of a hardware ceiling but because of a systems-level constraint: provisioning overhead, the “cold start” delay that precedes any encoding.

Previous studies have tackled cloud benchmarking [16], HPC workloads on cloud infrastructure [9], and multi-cloud scheduling [14], yet cost optimization across different instance families for media workloads remain underexplored. Comparing burstable, general-purpose, and compute-optimized tiers is tricky in its own right: burstable instances bring CPU credit mechanics and different pricing floors that existing benchmarks have not measured under realistic video encoding loads. We focus on H.264, H.265, and VP9, the three codecs that still dominate commercial pipelines; newer standards like VVC remain outside most production deployments owing to limited hardware acceleration and unresolved licensing on current EC2 generations.

We test AWS T, M, and C instance families, covering both x86 and ARM Graviton architectures, under three scaling strategies (serial, horizontal, vertical). Four questions drive the work:

- RQ1.** How do cost-performance profiles differ across instances families, CPU architectures, and codecs (H.264, H.265, VP9)?
- RQ2.** How does provisioning overhead constrain parallel encoding efficiency, and what fraction of total execution time does it represent?
- RQ3.** Can provisioning optimizations, specifically pre-built AMIs, make small-instance clusters competitive with or superior to vertical scaling, in both time and cost?
- RQ4.** How do regional pricing disparities and network egress costs define the economic floor of optimized VaaS deployments?

In summary, provisioning latency, rather than compute capacity, is the primary factor limiting horizontal scaling of short-lived transcoding jobs: in unoptimized deployments it accounted for 57.3%–93.3% of total wall-clock time across T-family configurations. A workload-optimized AMI reverses

In: Proceedings of the Brazilian Symposium on Multimedia and the Web (WebMedia’2026). Lavras/MG, Brazil. Porto Alegre: Brazilian Computer Society, 2026.

© 2026 SBC – Brazilian Computing Society.
ISSN 2966-2753

the economics: a 10-node **t3.micro** cluster outperforms a single **m5.4xlarge** by $1.28\times$ at 89.4% lower compute cost (62% after CPU credit surcharges), a speedup also observed intra-family ($10\times$ **m5.large** vs. **m5.4xlarge**), which rules out cross-family comparison artifacts. A chunk-size sweep and an egress cost model complete the analysis: once compute is optimized, data transfer becomes the dominant cost.

2 RELATED WORKS

The literature on cloud resource optimization for media workloads falls into two main strands: (i) generic cloud infrastructure benchmarking and provisioning, and (ii) work focused specifically on video encoding. Neither directly addresses the question examined here: the interaction between instance-family selection and provisioning strategy for repetitive, divisible encoding jobs. Infrastructure-oriented work tends to model workloads as monolithic tasks, whereas video-encoding studies mostly optimize throughput per monetary unit; for short-lived, horizontally distributed video chunks, provisioning latency becomes a hidden constant that both communities have largely overlooked.

On the infrastructure side, researchers have benchmarked cloud instances across workload classes [16], compared cloud and supercomputer execution for HPC [9], built cost-driven provisioning for scientific computing [1], and studied resource management [13] and thread scalability [17]. None measure the environment provisioning overhead that hurts horizontally scaled, chunk-based VaaS deployments.

On the encoding side, Moina-Rivera et al. [19] survey cloud and edge transcoding architectures, and Katsenou et al. [15] look at energy-codec trade-offs on embedded devices. Within the WebMedia community, recent encoding work targets codec-level optimizations such as fast intra-frame prediction heuristics for VVC and 360° video [7, 24]; we instead keep the encoder fixed and address the cloud provisioning layer around it. Jiang et al. [14] propose a multi-cloud cost-aware scheduler for big-data pipelines, but says little about video-specific workload properties. The closest predecessor to our parallelism approach is ExCamera [12] (Fouladi et al.), which reached 60–300 $\times$ speedups through fine-grained serverless (AWS Lambda) parallelism. We adopt the chunked-parallelism concept but pose a different question: what is the actual cost of ephemeral compute on EC2, where VM and environment provisioning introduce bottlenecks absent in serverless models, and where naïve parallelism can in fact degrade performance. AWS Elemental MediaConvert [3] avoids provisioning overhead entirely as a managed service, but its opaque pricing tiers preclude any instance-level control over margins.

Spot instance pricing [8, 27] can cut costs beyond what we report, but preemption risk makes it hard to guarantee the deterministic latency that VaaS SLAs require, so we leave that comparison for future work. Cloud performance variability is well-documented [2, 25]; this is why we ran 30 trials per configuration. Our chunked workload decomposition builds on

ideas from the fog-cloud [21] and distributed scheduling [26] literature.

3 EXPERIMENTAL METHODOLOGY

The study has four phases, each feeding into the next. Phase 1 characterizes the hardware environment; Phase 2 benchmarks raw encoding performance across the instance space; Phase 3 evaluates scaling strategies under realistic pipeline conditions; and Phase 4 quantifies how chunk duration and egress costs affect overall cost. All experiments were conducted on AWS in **us-east-1** (N. Virginia); selected configurations were replicated in **sa-east-1** (São Paulo) to assess regional price effects.

3.1 Phase 1: Validation of Processor Homogeneity

AWS does not guarantee that every instance of a given type runs on the same processor model. To measure this variability before running benchmarks, we launched 100 instances of each configuration (T, M, C families) across all six availability zones (a–f) in **us-east-1**, each running Ubuntu 22.04 LTS, and read `/proc/cpuinfo` on every one.

The results confirmed real heterogeneity, in line with prior cloud studies [10, 22, 23]: **t3.micro** in zone a was 98% Xeon 8259CL / 2% Xeon 8175M; **m5.large** in zone c split 60%/40% between 8259CL and 8175M; **c5.large** in zone b split 74%/26% between 8275CL and 8124M. ARM Graviton instances (**t4g**, **m7g**, **c7g**) were fully homogeneous (100%).

To keep the CPU model from confounding Phase 3 results, our provisioning scripts check `/proc/cpuinfo` as soon as SSH is up; if the instance landed on the wrong processor, it is terminated and relaunched until the target model is confirmed, so all 30 runs per configuration ended up on the same CPU model. Phase 3 ARM experiments use Graviton 3 only (**m7g.large**, **c7g.large**); we validated **t4g** (Graviton 2) but excluded it from scaling tests because its credit mechanics would confound the comparison.

3.2 Phase 2: Initial Performance Benchmarks

Phase 2 measures raw encoding performance across all 15 instance types with a containerized FFmpeg environment (H.264/AVC, H.265/HEVC, VP9).

The workload is a 320×240 (240p) test sequence with mixed motion. We chose this resolution deliberately: it keeps the bottleneck in the CPU and codec thread scheduler rather than in I/O or memory bandwidth, isolating instance-level compute differences that would get masked at 1080p or 4K by memory and disk pressure on smaller instances. Segments are 30 seconds long: sufficient for process initialization to be negligible relative to encoding time, yet short enough to permit 30 replications per configuration within a tractable total experiment time.

Each condition ran 30 times. Encoding used `ffmpeg -i input.mp4 -c:v <codec> -preset slow -b:v 2M -threads <n> output.mp4`. The `<codec>` parameter evaluated `libx264`, `libx265`, and `libvpx-vp9`; `<n>` was tested at 1, 3, 5, and 10

threads for all instances, plus 16 threads for **m5.4xlarge** and **c5.4xlarge** (the only 16-vCPU configurations). Encoding duration was recorded and cost calculated from instance pricing.

3.3 Phase 3: Horizontal vs. Vertical Scaling Analysis

Phase 3 builds on the instance ranking from Phase 2 to address a practical pipeline question: for a 10-minute video, is horizontal scaling (ten parallel small instances) preferable to vertical scaling (one large instance)? A 10-minute duration is long enough to require chunked parallel processing yet short enough to permit repeated end-to-end trials.

We compare three configurations across all T, M, and C families (x86 and ARM): horizontal ($10 \times$ small instance), vertical (single large instance), and serial ($1 \times$ small instance, full video).

The scaling investigation has two sub-phases. Phase 3.1 (Dynamic Provisioning) uses stock Ubuntu images with post-boot FFmpeg installation, the unoptimized baseline. Phase 3.2 (Optimized Provisioning) uses a pre-configured custom AMI with FFmpeg already installed, measuring the best case once runtime setup costs are gone. Both phases record end-to-end wall-clock time and total cost.

3.4 Phase 3 Extended Evaluation: ARM Graviton and Regional Pricing

We extended Phase 3 in two directions to test how well the findings generalize. First, two ARM Graviton 3 instances (**m7g.large** and **c7g.large**) were run alongside their x86 counterparts under the same horizontal and serial strategies, adding H.265 (**libx265**) and VP9 (**libvpx-vp9**) to probe codec-architecture interactions, with the same $n = 30$ protocol and terminate-and-retry CPU verification from Phase 1. Second, the horizontal H.264 evaluation was replicated in **sa-east-1** (São Paulo) for three representative instances (**t3.micro**, **m5.large**, **c5.large**) to assess how regional pricing changes the cost-performance picture.

3.5 Phase 4: Chunk Size Sensitivity and Cost Analysis

Phase 4 adds two analyses on top of the scaling results. First, a chunk size sensitivity analysis looks at how the segment duration c affects total wall-clock time Tc for a video of length L encoded across $N = \lceil Lc \rceil$ parallel instances, decomposed in Eq. 2. We tested three chunk durations: 30 s ($N = 20$), 60 s ($N = 10$), and 120 s ($N = 5$). Second, a network cost model captures the economic impact of data egress. Total pipeline cost C_{total} splits into compute provisioning cost $C_{\text{comp}} = \sum_{i=1}^N R_{\text{inst}} \times T_i$ and outbound transfer cost $C_{\text{net}} = R_{\text{egress}} \times D_{\text{out}}$, formalized in Eq. 3 and validated against empirical measurements in Section 4.5.

All execution times were recorded at millisecond resolution. Each configuration was repeated $n = 30$ times in both Phase 2 and Phase 3. With $n = 30$, the Central Limit Theorem

gives us a near-normal sampling distribution of the mean, so Welch's t -test is appropriate even when the raw data are not perfectly normal. We report descriptive statistics and 95% confidence intervals (Student's t -distribution) for every configuration. Statistical tests include Shapiro-Wilk for normality, Levene's for variance homogeneity, and Welch's t -tests for group comparisons.

4 EXPERIMENTAL RESULTS AND ANALYSIS

Results follow the four-phase structure laid out in Section 3.

4.1 Phase 1 Results: Processor Homogeneity Validation

Phase 1 (Section 3.1) confirmed real processor heterogeneity, in line with prior cloud benchmarking [22, 23]: T-family requests were effectively homogeneous, while M and C families split between two processor models, enough to confound latency comparisons. The terminate-and-relaunch protocol pinned every Phase 3 run to the dominant model of its family (8259CL for M-type, 8275CL for C-type).

4.2 Phase 2 Results: Initial Performance Benchmarks

Phase 2 benchmarks H.264, H.265, and VP9 across all 15 instance types at multiple thread counts, giving us the baseline cost-performance map that drives Phase 3 scaling decisions. Each configuration ran 30 times on isolated 30-second segments; we report compression time (seconds), efficiency, and total cost (Eq. 1).

4.2.1 Efficiency and Cost Calculation. These two metrics are inverses of each other, per Eq. 1:

$$\text{Total Cost} = \frac{\text{cost}h}{3600} \times \text{times}, \quad \text{Efficiency} = \frac{1}{\text{Total Cost}} \quad (1)$$

where $\text{cost}h$ is the hourly instance price from Table 1 and times is the measured encoding duration. All instance prices were retrieved in March 2026 via the AWS Pricing API.

4.2.2 Key Insights from Benchmark Data. Table 2 presents descriptive statistics for a representative subset of instances (the complete per-instance, per-thread table is in the artifact); it directly addresses RQ1 and reveals three patterns.

Cost-efficiency vs. raw speed. The **c5.4xlarge** is the fastest single instance (H.264, 10 threads: 1.1355 s), yet the **t3.micro** is substantially more cost-efficient: \$0.0000074 per operation vs. \$0.0002145, a $29 \times$ difference (efficiency 134,700 vs. 4,662, both at 10 threads, consistent with Table 2). Compute-optimized capacity buys speed rather than cost-efficiency.

Family hierarchy. The T-family leads in cost-efficiency across every codec and thread combination; the C-family leads in raw throughput at proportionally higher per-operation cost; the M-family lies in between.

Table 1: Experimental instance configurations and on-demand pricing (Linux, shared tenancy).

Instance	vCPU	RAM (GB)	\$/hr us-east-1	\$/hr sa-east-1
<i>Burstable (T-family)</i>				
t2.micro	1	1	0.0116	0.0186
t3.micro	2	1	0.0104	0.0168
t3.small	2	2	0.0208	0.0336
t3.medium	2	4	0.0416	0.0672
t3.large	2	8	0.0832	0.1344
t3.xlarge	4	16	0.1664	0.2688
t3.2xlarge	8	32	0.3328	0.5376
<i>General Purpose (M5-family, x86)</i>				
m5.large	2	8	0.0960	0.1530
m5.xlarge	4	16	0.1920	0.3060
m5.2xlarge	8	32	0.3840	0.6120
m5.4xlarge	16	64	0.7680	1.2240
<i>Compute Optimized (C5-family, x86)</i>				
c5.large	2	4	0.0850	0.1310
c5.xlarge	4	8	0.1700	0.2620
c5.2xlarge	8	16	0.3400	0.5240
c5.4xlarge	16	32	0.6800	1.0480
<i>ARM Graviton3 (M7g/C7g-family, aarch64)</i>				
m7g.large	2	8	0.0816	0.1301
c7g.large	2	4	0.0725	0.1111

Table 2: Descriptive statistics at 10 threads for a representative subset of instances.

Instance	H.264		H.265		VP9	
	Med.	Eff.	Med.	Eff.	Med.	Eff.
t2.micro	2.7602	112,434	4.8712	63,710	14.8775	20,860
t3.micro	2.5698	134,700	4.2868	80,749	17.4516	19,835
t3.2xlarge	1.5352	7,046	1.8224	5,936	17.2736	626
m5.large	2.4795	15,124	4.1685	8,996	17.1800	2,182
m5.4xlarge	1.3023	3,599	1.7620	2,660	16.8900	277
c5.large	2.1869	19,366	3.6379	11,642	15.0300	2,818
c5.4xlarge	1.1355	4,662	1.5050	3,517	14.5000	365

Thread scaling. All families exhibit diminishing returns beyond the instance’s vCPU count, consistent with thread-oversubscription effects [16]: immediately on the single-vCPU **t2.micro**, only past the thread limit on the 16-vCPU **c5.4xlarge**.

4.2.3 Statistical Analysis. Shapiro-Wilk and Levene’s tests flagged non-normality ($p < .05$ for M-family distributions) and unequal variances. That said, the coefficient of variation stayed low for most Phase 2 configurations, so the departures from normality are negligible in practice.

The compute-optimized **c5.large** was 11.8% faster than **m5.large** (10 threads, H.264; $p < .001$), consistent with its higher clock and cache architecture, while **t3.micro** was 18.9% slower than **c5.large** (1 thread, H.264) but $\approx 6.9\times$ more cost-efficient ($p < .001$). Because measurement variance is near-deterministic in containerized FFmpeg (CV $< 0.6\%$), we report practical significance through the absolute and percentage differences with 95% CIs rather than a standardized effect size, which the tiny denominator would inflate to uninformative magnitudes.

4.2.4 Visual Analysis of Results. Figure 1 plots median compression time at 10 threads for the representative x86 instances of Table 2, grouped by codec and colored by family. We report the median (consistent with Table 2) because it resists the occasional outlier and better captures central tendency under skew.

The pattern of diminishing returns in thread scaling is evident: the transition from 1 to 3 threads yields near-linear speedups on smaller instances (e.g., the **t3.micro**), but gains level off at 5 threads. Beyond that point, behavior is codec-dependent: H.264 keeps scaling through 16 threads on larger configurations ($\approx 34\text{--}38\%$ faster than at 10 threads), sustained by **libx264**’s intra-frame thread partitioning, whereas VP9 regresses significantly past 10 threads ($\approx 39\text{--}47\%$ slower at 16 threads). Both effects arise from thread oversubscription: once software threads exceed the physical cores, synchronization overhead offsets the gains from added parallelism.

4.3 Phase 3 Results: Horizontal vs. Vertical Scaling Analysis

Phase 2 showed **t3.micro** is the most cost-efficient configuration for H.264 encoding in isolation; Phase 3 tests whether that efficiency holds when a 10-minute video is encoded end-to-end, horizontally (10-node cluster) versus vertically (single large instance). The scope differs: Phase 2 measured raw encoding time on isolated 30-second segments (median ≈ 2.5 s on **t3.micro**), whereas Phase 3 measures full pipeline wall-clock time—provisioning, file transfer, encoding, and output retrieval—so its higher numbers (≈ 73 s per cluster run) include ≈ 48 s of provisioning overhead that Phase 2 never sees.

We use H.264 across all x86 families as the primary scaling benchmark to separate provisioning dynamics from codec-specific noise (Table 3); H.265 and VP9 come in for the ARM Graviton comparisons. For x86, each 10-node cluster ($10\times$ the smallest instance in a family) goes up against the largest single instance in that family: **t3.2xlarge**, **m5.4xlarge**, and **c5.4xlarge**. The nearest cost-equivalent vertical baseline for the **t3.micro** cluster (\$0.104/h) would be **t3.xlarge** (\$0.1664/h); we deliberately compare against the biggest nodes (up to \$0.768/h) instead, to test whether small-instance clusters can outperform, not just match, the highest-throughput option in each tier.

4.3.1 Two-Phase Analysis Results. Phase 3.1 used stock Ubuntu images with post-boot FFmpeg installation as an explicit worst-case baseline for RQ2—not a recommended practice, but the reference point that isolates provisioning cost. Post-boot dependency installation is a known anti-pattern; pre-warming compute to hide this latency is established practice, from EC2 Auto Scaling warm pools [5] to container reuse [11] and the cold-start mitigations studied in serverless video pipelines [12]. Our aim is therefore not to propose pre-warming but to quantify, rigorously and across instance families, how much of the cost picture it governs. Against this baseline, setup time consumed 57.3–93.3% of total wall-clock time, depending on configuration. Even so, vertical

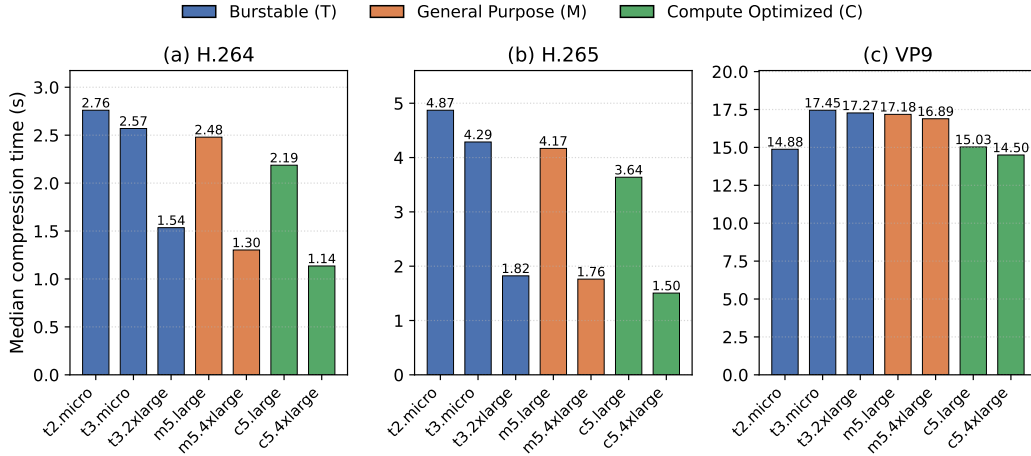


Figure 1: Median compression time at 10 threads ($n = 30$) across representative x86 instances for (a) H.264, (b) H.265, and (c) VP9.

scaling (t3.2xlarge, 3.04 min) edged out the horizontal cluster (t3.micro $\times$ 10, 3.28 min) by 7.3%, while the horizontal cluster won on cost: \$0.00569 versus \$0.01686, a 66.3% reduction despite being slower. Provisioning overhead is the binding factor, and quantifying its removal is what Phase 3.2 does.

Phase 3.2 replaced runtime dependency installation with pre-configured AMIs, targeting RQ3, and the improvement was immediate: the optimized t3.micro cluster completed in 1.22 min, 1.48 $\times$ faster than the vertical t3.2xlarge (1.80 min); the M-family horizontal cluster reached 1.22 min (1.28 $\times$ over the m5.4xlarge) and the C-family 1.18 min (1.26 $\times$ over the c5.4xlarge), against a serial t3.micro baseline of 2.72 min. In every family, the horizontal cluster was both faster and cheaper than the largest vertical instance once provisioning overhead was eliminated.

4.3.2 Analysis and Validation. Throughout, Setup Overhead (used interchangeably with provisioning latency) is the time from the EC2 run_instances API call until the SSH daemon responds; in parallel configurations, encoding time also covers SFTP segment upload and output retrieval. All experimental artifacts are at <https://github.com/breno-sh/webmedia2026-transcoding-artifact>.

Table 3 reports the full timing and cost data for both phases; Figure 2 shows how the gap between horizontal and vertical scaling closes once AMI optimization removes the provisioning bottleneck.

Within-family cost savings are modest in absolute terms (M-family horizontal saved 2.3%, C-family 1.0%): per-hour rates for smaller instances scale roughly proportionally to the larger ones, so the main within-family payoff of horizontal scaling is throughput, not cost.

The cost advantage is most pronounced in cross-family comparisons. Each t3.micro node processes its 60-second

chunk in a single burst well within the credit window, delivering 100% vCPU throughput during encoding. Under these conditions, the t3.micro cluster (\$0.00211) costs 89.4% less than the m5.4xlarge (\$0.01997) and 87.5% less than the c5.4xlarge (\$0.01689), while completing faster than both.

Dynamic Phase 3.1 data for 10 $\times$ m5.large are excluded from Table 3: all 30 runs encountered a systematic FFmpeg muxer overflow (“too many packets buffered”) on segments 1 and 4, caused by a known bug in the FFmpeg version from the Ubuntu 20.04 apt repository; no complete encoding succeeded. The bug does not appear in Phase 3.2, whose pre-built AMI ships a patched FFmpeg, a further sign that AMI optimization matters beyond raw provisioning speed.

4.3.3 Methodological and Statistical Validation. Welch’s t -tests confirm the Phase 3 speedups are real, not variance artifacts: $p < 10^{-23}$ for T-family horizontal vs. vertical, and $p < 10^{-44}$ for both M-family and C-family. ARM Graviton encoding variance is low once transfer artifacts are factored out: all reconstructed configurations in Table 4 achieve CV<10%, with provisioning itself measured at CV= 1.9% ($n = 100$), more stable than x86 (CV= 4.4%).

The intra-family comparisons (10 $\times$ m5.large vs. m5.4xlarge and 10 $\times$ c5.large vs. c5.4xlarge) are the cleanest test of the horizontal scaling hypothesis. Both sides are in the same instance family and billing tier, which removes confounds from CPU credit mechanics, cross-family pricing gaps, and heterogeneous workload behavior. The 1.28 $\times$ and 1.26 $\times$ speedups in M and C families confirm that, under the workload and provisioning conditions evaluated here, the horizontal-scaling advantage is architectural rather than specific to burstable instances or an artifact of comparing burstable against non-burstable tiers.

Table 3: Scaling strategies across optimization phases (Dynamic Ubuntu vs. Optimized pre-built AMI). Speedup is relative to the serial baseline of each family and provisioning phase (serial rows are $1.00\times$ by definition); Cost $\times$ is total cost relative to the serial AMI baseline. ARM results are reported in Table 4.

Instance	Prov.	Time (min)	Cost (USD)	Speedup	Cost $\times$
<i>Burstable (T)</i>					
t3.micro	Dynamic	4.03 ± 0.12	\$0.00070	$1.00\times$	$1.0\times$
t3.micro	Optimized	2.72 ± 0.08	\$0.00047	$1.00\times$	$1.0\times$
$10\times$ t3.micro	Dynamic	3.28 ± 0.10	\$0.00569	$1.23\times$	$8.1\times$
$10\times$ t3.micro	Optimized	1.22 ± 0.04	\$0.00211	$2.23\times$	$4.5\times$
t3.2xlarge	Dynamic	3.04 ± 0.09	\$0.01686	$1.33\times$	$24.1\times$
t3.2xlarge	Optimized	1.80 ± 0.05	\$0.00998	$1.51\times$	$21.2\times$
<i>General Purpose (M)</i>					
m5.large	Dynamic	3.46 ± 0.10	\$0.00554	$1.00\times$	$1.0\times$
m5.large	Optimized	2.71 ± 0.08	\$0.00434	$1.00\times$	$1.0\times$
$10\times$ m5.large	Optimized	1.22 ± 0.04	\$0.01952	$2.22\times$	$4.5\times$
m5.4xlarge	Dynamic	2.33 ± 0.07	\$0.02982	$1.48\times$	$5.4\times$
m5.4xlarge	Optimized	1.56 ± 0.05	\$0.01997	$1.74\times$	$4.6\times$
<i>Compute Optimized (C)</i>					
c5.large	Dynamic	3.21 ± 0.10	\$0.00455	$1.00\times$	$1.0\times$
c5.large	Optimized	2.46 ± 0.07	\$0.00349	$1.00\times$	$1.0\times$
$10\times$ c5.large	Dynamic	3.52 ± 0.11	\$0.04987	$0.91\times$	$11.0\times$
$10\times$ c5.large	Optimized	1.18 ± 0.04	\$0.01672	$2.08\times$	$4.8\times$
c5.4xlarge	Dynamic	2.13 ± 0.06	\$0.02414	$1.51\times$	$5.3\times$
c5.4xlarge	Optimized	1.49 ± 0.04	\$0.01689	$1.65\times$	$4.8\times$

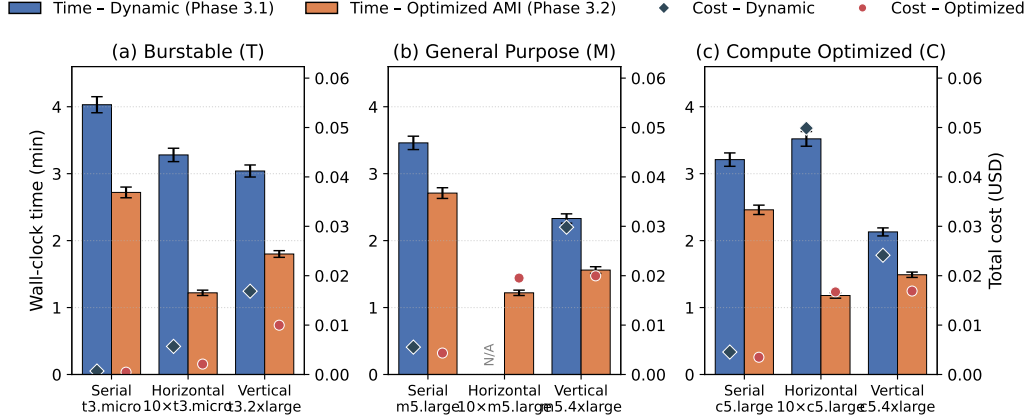


Figure 2: Performance and cost scaling across (a) T, (b) M, and (c) C instance families under Dynamic (Phase 3.1) and Optimized-AMI (Phase 3.2) provisioning. Bars denote wall-clock time (95% CI, $n = 30$); markers denote total cost.

4.4 Architecture, Codec, and Regional Impact Analysis

The ARM Graviton and regional-pricing extensions of Phase 3 test whether the x86 findings transfer to a different ISA and whether they hold outside `us-east-1`.

ARM Graviton Performance (Codecs). Running the `c7g.large` and `m7g.large` instances alongside their x86 counterparts revealed notable interactions between architecture and codec. Table 4 reports reconstructed times (encoding plus the median provisioning latency of 20.5s, CV= 1.9%, from an independent $n = 100$ characterization, factoring out SFTP transfer artifacts). For H.264, `m7g.large` horizontal scaling gave a $1.72\times$ speedup over serial (33.55s vs. 57.55s), while

`c7g.large` showed no horizontal benefit at all (107.68s horizontal $\approx$ 107.41s serial). For H.265 the relationship was reversed: `c7g.large` horizontal clusters (61.03s) substantially outperformed the `m7g.large` (182.01s), a $2.98\times$ difference between two ARM instance types. A comparison of ARM H.265 against x86 H.264 would be misleading, since the codecs differ in computational complexity.

The high H.265 serial times (≈ 700 s on both instances) reflect the computational intensity of `libx265 -preset slow` on ARM: current `libx265` builds lack the SIMD optimization depth available on x86 [20]; the same 10-minute H.265 encoding on x86 (`c5.large` serial) finishes in ≈ 72 s, an order of magnitude faster.

Two H.265 serial rows in Table 4 use an adjusted n : isolated container orchestration anomalies caused complete pipeline

stalls in a handful of runs, producing extreme outliers (>5 standard deviations from the mean), which we excluded ($n = 26$ for **m7g.large**, $n = 29$ for **c7g.large**). The resulting **m7g.large** H.265 serial distribution is unimodal with low variance (median 700.51 s, stdev 4.10 s).

Statistical Validation: Welch's t -test found no significant difference between **m7g.large** and **c7g.large** for H.265 horizontal encoding ($p > .05$), with moderate variability (CV 4.5–6.2%) in H.265 horizontal runs; a Mann-Whitney U test agreed ($p > .05$). Welch's t -test did confirm the H.264 inversion on **m7g.large** at high confidence ($p < 10^{-12}$): this is not noise but a structural effect of provisioning overhead dominating encoding time at this chunk granularity.

Regional Cost Disparities. Running the same experiments in **sa-east-1** (São Paulo) revealed a 54–62% price premium over **us-east-1**, varying by family (T: +61.5%, M: +59.4%, C: +54.1%). Applying that premium to the optimized 10-node H.264 clusters of Table 3 (**us-east-1**: \$0.00211, \$0.01952, \$0.01672 for **t3.micro**, **m5.large**, **c5.large**) yields **sa-east-1** costs of \$0.00341, \$0.03112, and \$0.02576, respectively—a 1.5–1.6 $\times$ increase. Wall-clock times stayed the same across regions: compute performance is hardware-bound, and the cost difference is purely a pricing artifact. Beyond cost, regional placement also affects user-facing quality-of-experience metrics such as delivery latency and time-to-first-frame, which our infrastructure-side measurements do not capture; quantifying that trade-off is left for future work.

4.5 Phase 4 Results: Chunk Size Sensitivity and Cost Analysis

Phase 4 adds two analyses that address RQ4: an empirical study of chunk duration sensitivity and a cost model for network egress.

Chunk Size Sensitivity. For a video of length L processed across $N = \lceil Lc \rceil$ parallel instances, total wall-clock time depends on the chunk duration c in a non-monotone way:

$$Tc = t_{\text{seg}} \max_{1 \leq i \leq N} (t_{\text{prov},i}) t_{\text{enc}} t_{\text{merge}} \quad (2)$$

Shorter chunks mean more instances (N goes up), which widens the expected maximum provisioning time, the synchronization barrier at the end of the parallel phase; longer chunks mean fewer workers and less parallelism, making encoding time the bottleneck. We measured this trade-off on the 10-minute H.264 workload with the Phase 3.2 AMI across all three x86 families. Total Time in Table 6 is end-to-end wall-clock; speedup is relative to each family's vertical baseline from Table 3.

For 240p content, 60-second chunks were optimal across the board: 30-second chunks hurt performance because 20 provisioning sequences widen the long-tail outlier; 120-second chunks left only 5 workers, not enough parallelism.

Resolution Dependence. To check whether this optimum holds beyond low-resolution workloads, we replicated the M-family H.264 chunk-sensitivity experiment at 720p and 1080p

on **m5.large** horizontal clusters (Table 5; boldface marks the fastest chunk size per resolution). The optimal chunk size shifts with resolution: at 240p, where per-chunk encoding (≈ 5 s) is dominated by provisioning (≈ 18 s), 60-second chunks balance parallelism against provisioning overhead. At 720p and 1080p, encoding time grows by 10 $\times$ and 26 $\times$, respectively, so parallelism matters more: 30-second chunks become optimal because the extra provisioning cost is amortized over much longer per-chunk encoding. SFTP transfer overhead also scales with chunk file size, adding 26–199 s per chunk at higher resolutions; an S3-based pipeline would shrink this, but was not tested. Because the resolution sweep used a single trial per configuration, we report these values as indicative rather than conclusive; repeating the sweep with $n = 30$ and confidence intervals is left for future work.

The Network Egress Economic Bottleneck. A complete pipeline cost model must also capture data egress: the total cost (C_{total}) is the sum of compute provisioning (C_{comp}) and outbound data transfer (C_{net}):

$$C_{\text{total}} = \left(\sum_{i=1}^N R_{\text{inst}} \times T_i \right) (R_{\text{egress}} \times D_{\text{out}}) \quad (3)$$

where R_{inst} is the per-second instance billing rate, T_i is the active billing lifespan of node i ,¹ R_{egress} is the regional outbound internet data transfer rate (e.g., \$0.09/GB in **us-east-1**, \$0.25/GB in **sa-east-1**), and D_{out} is the final payload size in GB.

Intra-region transfers carry no per-GB charge; outbound internet egress, however, sets a hard economic floor: delivering the ≈ 100 MB H.264 output costs $\approx \$0.009$ in **us-east-1** ($\approx 4\times$ the optimized **t3.micro** compute cost of \$0.00211), which makes egress the dominant cost once compute is optimized.

Routing delivery through a CDN (S3 + CloudFront) cuts this by $\approx 10\times$: CloudFront charges \$0.0085/GB vs. \$0.09/GB for direct egress, making CDN delivery cheaper than compute above ≈ 2 GB/month of output.

5 DISCUSSION AND THREATS TO VALIDITY

AMI vs. runtime-install overhead. In Phase 3.2 all dependencies are baked into the AMI, so the residual ≈ 48 s overhead is just EC2 launch, EBS attach, boot, and SSH readiness, whereas Phase 3.1 installs FFmpeg post-boot via **apt**; the two phases are not interchangeable baselines.

Burstable CPU credit surcharges. T-family instances run in Unlimited mode [4], drawing surplus credits from launch; our ephemeral spawn-encode-terminate pattern never idles long enough to offset that draw. We bound the effect: at a realistic ≈ 22 s of full-CPU encoding the per-cluster surcharge is $\approx \$0.0055$ (cluster cost $\approx \$0.0076$, still 62% below **m5.4xlarge**); in the worst case (100% CPU for the full 73 s) it reaches $\approx \$0.0183$ and erases the advantage.

¹AWS bills a 60-second minimum per instance; all observed T_i exceed it.

Table 4: Descriptive statistics (compression time, efficiency, and total cost) for the ARM Graviton configurations.

Instance	Strategy	H.264			H.265 (libx265)			VP9 (libvpx-vp9)		
		Median	Std	CV(%)	Median	Std	CV(%)	Median	Std	CV(%)
m7g.large	serial	57.55	0.61	1.1	700.51†	4.10	0.6	329.03	1.41	0.4
m7g.large	horizontal	33.55	3.04	8.8	182.01	11.39	6.2	63.10	2.53	4.0
c7g.large	serial	107.41	1.79	1.7	702.15†	1.36	0.2	328.75	1.60	0.5
c7g.large	horizontal	107.68	18.34	17.0	61.03	2.76	4.5	63.27	6.02	9.3

†H.265 serial rows use reduced n ($n = 26$ for **m7g.large**, $n = 29$ for **c7g.large**) after outlier removal; all others $n = 30$.

Table 5: Chunk-size sensitivity across resolutions.

Resolution	Chunk	Total (min)	Avg Encode (s)	Avg Upload (s)
<i>240p (320 × 240)</i>				
	30 s	1.63	≈5	<1
	60 s	1.22	≈5	<1
	120 s	2.10	≈5	<1
<i>720p (1280 × 720)</i>				
	30 s	4.12	50.6	66.2
	60 s	5.31	95.5	128.6
	120 s	7.73	199.9	135.1
<i>1080p (1920 × 1080)</i>				
	30 s	5.49	129.8	21.9
	60 s	6.40	219.7	48.6
	120 s	9.00	371.7	52.0

Table 6: Chunk size sensitivity: wall-clock time and speedup over vertical baseline.

Family	Chunk Strategy	Total Time (min)	Speedup vs Vertical
<i>Burstable (T)</i>			
	30 s (20 × t3.micro)	1.39	1.30×
	60 s (10 × t3.micro)	1.22	1.48×
	120 s (5 × t3.micro)	2.04	0.88×
<i>General (M)</i>			
	30 s (20 × m5.large)	1.63	0.96×
	60 s (10 × m5.large)	1.22	1.28×
	120 s (5 × m5.large)	2.10	0.74×
<i>Compute (C)</i>			
	30 s (20 × c5.large)	1.44	1.03×
	60 s (10 × c5.large)	1.18	1.26×
	120 s (5 × c5.large)	1.98	0.75×

* Speedup < 1.0× indicates horizontal cluster is slower than vertical.

Alternative baselines. We tested on-demand EC2 with custom orchestration. AWS-native services (Elemental Media-Convert [3], Fargate) get near-zero provisioning overhead by design; spot instances [8, 27] are cheaper but carry preemption risk. Both comparisons are left for future work.

Transfer overhead. We used SFTP to distribute chunks within a single AZ. Horizontal configurations transfer N chunks simultaneously from the orchestrating host, while vertical transfers just one file, so SFTP penalizes horizontal scaling relative to an S3-based pipeline; the speedup advantages we report are conservative lower bounds.

Content dependence. Encoding time is content-dependent, and our per-instance benchmarks use a single mixed-motion test sequence at 240p. Higher-complexity content would lengthen the encoding term $t_{\text{enc}c}$ relative to provisioning, shifting the crossover chunk size downward, but it does not change the qualitative finding that provisioning dominates short divisible jobs. Broader content sets are needed before the specific chunk optima are treated as general.

Single-provider scope. All experiments ran on AWS EC2, so the absolute pricing figures reflect one provider’s billing mechanics and should not be read as cross-cloud constants. The structural result—that provisioning latency, not CPU throughput, governs the cost of short-lived horizontally-scaled transcoding—depends only on the existence of a non-trivial VM cold start, common to IaaS providers; the numbers, however, must be re-derived per provider, and a multi-cloud replication is left for future work.

Scope. ARM homogeneity was validated in **us-east-1** only. All shared-tenancy instances are subject to noisy-neighbor effects (mitigated by $\text{CV} < 0.6\%$ and $n = 30$). Our recommendations hold for the evaluated workloads and conditions; pricing and hardware generations change, so absolute figures should be re-validated before production use.

6 CONCLUSION

For short-form video encoding on EC2, the dominant cost driver is not the CPU, but the latency of waiting for instances to become ready. For RQ1, cost-performance profiles vary sharply across families: the T-family leads in cost-efficiency, the C-family in raw throughput, and VP9 remains the most expensive codec tested. For RQ2, unoptimized provisioning overhead consumed 57.3–93.3% of wall-clock time, canceling the benefit of horizontal scaling. For RQ3, workload-optimized AMIs reversed this picture: a 10-node **t3.micro** cluster outperformed a single **m5.4xlarge** by 1.28× at 89.4% lower compute cost, a result that held within instance families and extended to ARM Graviton H.265 throughput. For RQ4, the optimal chunk size proved resolution-dependent and, once compute was optimized, outbound egress became the binding constraint. Eliminating provisioning overhead is therefore the prerequisite for cost-effective horizontal cloud transcoding; GPU-accelerated instances and spot pricing are the most promising next steps.

ETHICAL CONSIDERATIONS

This study did not involve human participants, personal data, or any biological material, and therefore did not require review by a Research Ethics Committee under Resolution 510/2016 of the Brazilian National Health Council (CEP/CONEP system). All measurements were collected by the authors from cloud infrastructure that they provisioned and controlled: instance provisioning latencies, encoding times, and billed costs on Amazon EC2. The video content used in the experiments is a synthetic mixed-motion test sequence containing no identifiable individuals or private information. The public research artifact contains only aggregated performance and cost data together with the benchmarking and analysis scripts; it includes no account identifiers, access credentials, IP addresses, or any other personally identifiable or provider-confidential information. The reported pricing reflects publicly listed AWS on-demand rates and does not disclose contractual or account-specific terms. The authors declare no conflicts of interest.

ACKNOWLEDGEMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior — Brasil (CAPES) — Finance Code 001.

We also acknowledge the use of AI-based tools in the preparation of this article. Large language model assistants were used to support English-language writing and proof-reading, to help organize and format tables and figures, and to assist with data-analysis and plotting scripts. All experimental design, execution, measurements, statistical analysis, interpretation, and conclusions are the authors' own; every AI-assisted output was reviewed, verified, and edited by the authors, who take full responsibility for the content of this paper.

REFERENCES

- [1] P. Álvarez, S. Hernández, J. Fabra, and J. Ezpeleta. 2018. Cost-driven provisioning and execution of a computing-intensive service on the Amazon EC2. *Comput. J.* 61, 9 (2018), 1407–1421. doi:10.1093/comjnl/bxy006
- [2] O. Alzakholi, L. M. Haji, H. M. Shukur, R. R. Zebari, S. M. Abas, and M. Sadeeq. 2020. Comparison Among Cloud Technologies and Cloud Performance. *J. Appl. Sci. Technol. Trends* 1, 1 (2020), 40–47. doi:10.38094/jastt1219
- [3] Amazon Web Services. 2024. AWS Elemental MediaConvert. AWS Documentation. [Online]. Available: <https://aws.amazon.com/mediaconvert/> [Accessed: Mar. 2026].
- [4] Amazon Web Services. 2024. Burstable performance instances. Amazon EC2 User Guide. [Online]. Available: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-performance-instances.html> [Accessed: Mar. 2026].
- [5] Amazon Web Services. 2024. Warm pools for Amazon EC2 Auto Scaling. Amazon EC2 Auto Scaling User Guide. [Online]. Available: <https://docs.aws.amazon.com/autoscaling/ec2/userguide/ec2-auto-scaling-warm-pools.html> [Accessed: Aug. 2026].
- [6] Amazon Web Services. 2026. Amazon EC2 Instance Types. <https://aws.amazon.com/ec2/instance-types/>. Accessed Jun. 2026.
- [7] L. Araújo, A. Duarte, G. Correa, B. Zatt, and D. Palomino. 2024. Fast ISP Mode Decision for the Versatile Video Coding Intra Prediction Using Machine Learning. In *Proc. 30th Brazilian Symp. Multimedia Web*. SBC, Juiz de Fora, Brazil, 162–170. doi:10.5753/webmedia.2024.243129
- [8] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafir. 2013. Deconstructing Amazon EC2 Spot Instance Pricing. *ACM Trans. Econ. Comput.* 1, 3 (2013), 16:1–16:20. doi:10.1145/2509413.2509416
- [9] T. Dancheva, U. Alonso, and M. D. Barton. 2023. Cloud benchmarking and performance analysis of an HPC application in Amazon EC2. *Cluster Computing* 27, 2 (2023), 2273–2290. doi:10.1007/s10586-023-04060-4
- [10] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowlers, and M. M. Swift. 2012. More for Your Money: Exploiting Performance Heterogeneity in Public Clouds. In *Proc. 3rd ACM Symp. Cloud Comput.* ACM, San Jose, CA, USA, 20:1–20:14. doi:10.1145/2391229.2391249
- [11] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. 2015. An Updated Performance Comparison of Virtual Machines and Linux Containers. In *2015 IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*. Philadelphia, PA, USA, 171–172. doi:10.1109/ISPASS.2015.7095802
- [12] S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalariao, A. Sivaraman, G. Porter, and K. Winstein. 2017. Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads. In *Proc. 14th USENIX Symp. Networked Systems Design and Implementation (NSDI)*. USENIX Association, Boston, MA, USA, 363–376. [Online]. Available: <https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi> [Accessed: Mar. 2026].
- [13] L. M. Haji, S. R. M. Zeebaree, O. M. Ahmed, A. B. Sallow, K. Jacksi, and R. R. Zeabri. 2020. Dynamic Resource Allocation for Distributed Systems and Cloud Computing. *Test Engineering and Management* 83, 1 (2020), 22417–22426.
- [14] F. Jiang, K. Ferriter, and C. Castillo. 2020. A Cloud-Agnostic Framework to Enable Cost-Aware Scheduling of Applications in a Multi-Cloud Environment. In *2020 IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, Budapest, Hungary, 1–9. doi:10.1109/NOMS47738.2020.9110325
- [15] A. Katsenou, X. Wang, D. Schien, and D. Bull. 2024. Comparative Study of Hardware and Software Power Measurements in Video Compression. In *2024 Picture Coding Symp. (PCS)*. IEEE, Taichung, Taiwan, 1–5. doi:10.1109/PCS60826.2024.10566286
- [16] W. Maas, F. Rossi, M. Caggiani, P. Navaux, and A. Lorenzon. 2025. Investigating cloud instances to achieve optimal trade-offs between performance-cost efficiency. *Computing* 107, 1 (2025), 82–97. doi:10.1007/s00607-025-01444-9
- [17] W. Maas, F. D. Rossi, M. C. Luizelli, P. O. A. Navaux, and A. F. Lorenzon. 2025. Towards Optimizing Cost and Performance for Parallel Workloads in Cloud Computing. In *Proc. 15th Int. Conf. Cloud Comput. and Services Science (CLOSER)*. SciTePress, Porto, Portugal, 231–238. doi:10.5220/0013421600003950
- [18] A. Mahida. 2022. Comprehensive Review on Optimizing Resource Allocation in Cloud Computing for Cost Efficiency. *J. Artif. Intell. Cloud Comput.* 1, 2 (2022), 1–4. doi:10.47363/JAICC/2022(1)232
- [19] W. Moína-Rivera, M. García-Pineda, J. Gutiérrez-Aguado, and J. M. Alcaraz-Calero. 2024. Cloud media video encoding: review and challenges. *Multimedia Tools and Applications* 83 (2024), 81231–81278. doi:10.1007/s11042-024-18763-2
- [20] MulticoreWare Inc. 2023. x265 HEVC Encoder Documentation: Architecture-specific Optimizations. x265 Project Documentation. [Online]. Available: <https://x265.readthedocs.io/en/master/> [Accessed: Mar. 2026].
- [21] T. S. Nikoui, A. Balador, A. M. Rahmani, and Z. Bakhshi. 2020. Cost-Aware Task Scheduling in Fog-Cloud Environment. In *2020 CSI/CPSSI Int. Symp. Real-Time Embedded Syst. Technol. (RTEST)*. IEEE, Tehran, Iran, 1–8. doi:10.1109/RTEST49666.2020.9140118
- [22] Z. Ou, H. Zhuang, A. Lukyanenko, J. K. Nurminen, P. Hui, V. V. Mazalov, and A. Ylä-Jääski. 2013. Is the Same Instance Type Created Equal? Exploiting Heterogeneity of Public Clouds. *IEEE Trans. Cloud Comput.* 1, 2 (2013), 201–214. doi:10.1109/TCC.2013.12
- [23] Z. Ou, H. Zhuang, J. K. Nurminen, A. Ylä-Jääski, and P. Hui. 2012. Exploiting Hardware Heterogeneity within the Same Instance Type of Amazon EC2. In *Proc. 4th USENIX Conf. Hot Topics Cloud Comput.* USENIX Association, Boston, MA, USA, 4–4.
- [24] R. Rosado, O. Santos, F. Oliveira, L. Silva, G. Sanchez, I. Storch, D. Palomino, and L. Agostini. 2024. A Coding-Efficiency-Aware Fast Heuristic for VVC Intra-Frame Prediction Targeting 360° Videos. In *Proc. 30th Brazilian Symp. Multimedia Web*. SBC, Juiz de Fora, Brazil, 355–359. doi:10.5753/webmedia.2024.243128
- [25] J. Scheuner and P. Leitner. 2018. Estimating Cloud Application Performance Based on Micro-Benchmark Profiling. In *2018 IEEE*

- 11th Int. Conf. Cloud Comput. (CLOUD)*. IEEE, San Francisco, CA, USA, 90–97. doi:10.1109/CLOUD.2018.00019
- [26] Y. Yang and H. Casanova. 2003. RUMR: Robust Scheduling for Divisible Workloads. In *Proc. 12th IEEE Int. Symp. High Perform. Distrib. Comput.* IEEE, Seattle, WA, USA, 114–123. doi:10.1109/HPDC.2003.1210021
- [27] A. Zabrovskiy, P. Agrawal, V. Kashansky, R. Kersche, C. Timmerer, and R. Prodan. 2022. FSpot: Fast and Efficient Video Encoding Workloads Over Amazon Spot Instances. *Computers, Materials & Continua* 71, 3 (2022), 5677–5694. doi:10.32604/cmc.2022.023630